MOTOROLA

Embedded SDK (Software Development Kit)

V.8bis Library

SDK120/D
Rev. 2, 07/23/2002

MOTOROLA intelligence everywhere™ | digital dna™

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

Contents

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

List of Tables

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

List of Figures

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

List of Examples

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

About This Document

This manual describes the V.8bis telecommunication algorithm for use with Motorola's Embedded Software Development Kit (SDK).

Audience

This document targets software developers implementing communications (primarily, modem) functions within software applications.

Organization

This manual is arranged in the following sections:

- **Chapter 1, Introduction**—provides a brief overview of this document
- **Chapter 2, Directory Structure**—provides a description of the required core directories
- **Chapter 3, V.8bis Library Interfaces**—describes all of the V.8bis Library functions
- **Chapter 4, Building the V.8bis Library**—tells how to execute the system library project build
- **Chapter 5, Linking Applications with the V.8bis Library**—describes the organization of the V.8bis Library
- **Chapter 6, V.8bis Applications**—describes the use of V.8bis Library through test/demo applications
- **Chapter 7, License**—provides the license required to use this product
- **Appendix A, Guidelines for Setting Up the Input Buffer**—assists in setting up the input buffer

Suggested Reading

We recommend that you have a copy of the following references:

- *DSP56800 Family Manual*, DSP56800FM/AD
- *DSP568xx User's Manual* for the DSP device being implemented
- *Inside CodeWarrior: Core Tools*, Metrowerks Corp.

Conventions

This document uses the following notational conventions:

Typeface, Symbol or Term	Meaning	Examples
`Courier Monospaced Type`	Commands, command parameters, code examples, expressions, datatypes, and directives	...`*Foundational include files`... ...a data structure of type `vad_tConfigure`...
Italic	Calls, functions, statements, procedures, routines, arguments, file names and applications	...the *pConfig* argument... ...defined in the C header file, *aec.h*... ...makes a call to the *Callback* procedure...
Bold	Reference sources, paths, emphasis	...refer to the **Targeting DSP56824 Platform** manual.... ... see: **C:\Program Files\Motorola\Embedded SDK\help\tutorials**
Bold/Italic	Directory name, project name	...and contains these core directories: ***applications*** contains applications software.... ...CodeWarrior project, ***3des.mcp***, is.....
Blue Text	Linkable on-line	...refer to **Chapter 7**, License...
Number	Any number is considered a positive value, unless preceded by a minus symbol to signify a negative value	3V -10 DES^{-1}
ALL CAPITAL LETTERS	Variables, directives, defined constants, files libraries	INCLUDE_DSPFUNC #define INCLUDE_STACK_CHECK
Brackets [...]	Function keys	...by pressing function key [F7]...
Quotation marks “... “	Returned messages	...the message, “Test Passed” is displayed.... ...if unsuccessful for any reason, it will return “NULL”....

Definitions, Acronyms, and Abbreviations

The following list defines the acronyms and abbreviations used in this document. As this template develops, this list will be generated from the document. As we develop more group resources, these acronyms will be easily defined from a common acronym dictionary. Please note that while the acronyms are in solid caps, terms in the definition should be initial capped ONLY IF they are trademarked names or proper nouns.

ACK Acknowledge Message

CL Capabilities List

CLR Capabilities List Request

CR Capabilities Request

DCE Data Circuit-terminating Equipment

DTE	Data Terminal Equipment
DSP	Digital Signal Processor or Digital Signal Processing
ES	Escape Signal
FCS	Frame Check Sequence
I/O	Input/Output
IDE	Integrated Development Environment
LSB	Least Significant Bit
MAC	Multiply/Accumulate
MIPS	Million Instructions Per Second
MR	Mode Request
MS	Mode Select
MSB	Most Significant Bit
NAK	Negative Acknowledge Message
OnCE™	On-Chip Emulation
OMR	Operating Mode Register
PC	Program Counter
PSTN	Public Switched Telephone Network
SDK	Software Development Kit
SP	Stack Pointer
SPI	Serial Peripheral Interface
SR	Status Register
SRC	Source

References

The following sources were referenced to produce this book:

1. DSP56800 Family Manual, DSP56800FM/AD
2. DSP568xx User's Manual
3. Embedded SDK Programmer's Guide
4. ITU-T V.8bis Standard, Revision 1998

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

Chapter 1
Introduction

Welcome to Motorola's family of Digital Signal Processors, DSPs. This document describes the V.8bis Library, which is a part of Motorola's comprehensive Software Development Kit, SDK, for its DSPs. In this document, you will find all the information required to use and maintain the V.8bis Library interface and algorithms.

Motorola provides these algorithms to you for use with Motorola DSPs to expedite your application development and reduce the time it takes to bring your own products to market.

Motorola's V.8bis Library is licensed for your use on Motorola processors. Please refer to the standard Software License Agreement in **Chapter 7** for license terms and conditions; please consult with your Motorola representative for premium product licensing.

1.1 Quick Start

Motorola's Embedded SDK is targeted to a large variety of hardware platforms. To take full advantage of a particular hardware platform, use **Quick Start** from the **Targeting DSP568xx Platform** documentation.

For example, the **Targeting DSP56824 Platform** manual provides more specific information and examples about this hardware architecture. If you are developing an application for the DSP56824EVM board, or any other DSP56824 development system, refer to the **Targeting DSP56824 Platform** manual for **Quick Start** or other DSP56824-specific information.

1.2 Overview of V.8bis

The V.8bis allows DCEs and DTEs with multiple modes of operation over the PSTN and on leased telephone-type circuits, to perform these functions:

- selection of the desired mode of operation at automatic call establishment on the PSTN, controlled by either the calling or answering station
- selection of the desired mode of operation while in telephony mode on an already-established connection, controlled by either station
- determination by either station of whether the remote station supports V.8bis, with minimum disturbance to a voice caller
- exchange of available capabilities between stations on a connection at call establishment or while in telephony mode
- graceful recovery in the event of transmission of errors or selection of an unavailable mode of operation

These functions are provided by defining a set of signals, messages and procedures. Signals are intended to be detected in the presence of an interfering voice or other audio; to turn around any echo suppressors in the network prior to the beginning of information transmission and to indicate the initiation of a V.8bis transaction to the receiving station, while not appearing to the user and receiver as an indication of a data or facsimile device.

Messages convey significantly more information than signals. Messages can only be used when they will not cause disruption to a voice caller. They are intended to be used only in the absence of an interfering voice or other audio.

This Recommendation provides for error detection and rejection of corrupted messages, and rejection of mode selections that are unavailable.

1.2.1 Background

Two stations, Initiating and Responding, can have different capabilities, some of which are common. A common mode of communication must be agreed upon by both sides, depending on the priorities on each side. This is facilitated by V.8bis recommendation.

1.2.2 Features and Performance

The V.8bis library is not multichannel.

For details on Memory and MIPS for a particular DSP, refer to the **Libraries** chapter of the appropriate Targeting manual.

Chapter 2
Directory Structure

2.1 Required Core Directories

Figure 2-1 details required platform directories:

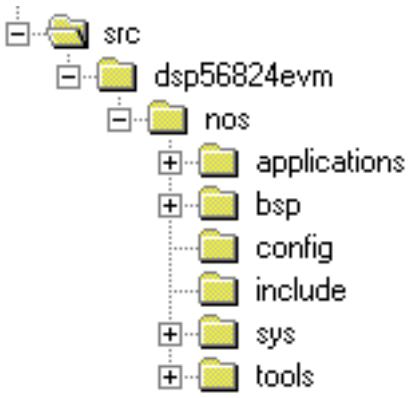

Figure 2-1. Core Directories

As shown in **Figure 2-1**, DSP56824EVM has no operating system support (nos), and includes the following core directories:

- ***applications*** contains applications software that can be exercised on this platform
- ***bsp*** contains board support package specific for this platform
- ***config*** contains default HW/SW configurations for this platform
- ***include*** contains SDK header files which define the Application Programming Interface
- ***sys*** contains required system components
- ***tools*** contains useful utilities used by system components

There are also optional directories that include domain-specific libraries.

2.2 Optional (Domain-Specific) Directories

Figure 2-2 demonstrates how V.8bis is encapsulated in the domain-specific directory, ***modem***.

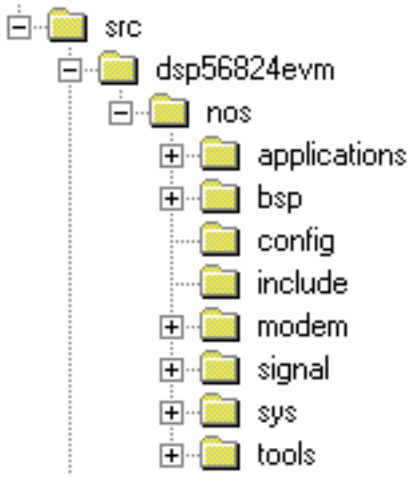

Figure 2-2. *Modem* Directory

The ***modem*** directory includes modem-specific algorithms, such as V.8bis, V.22bis and others. The ***V.8bis*** directory includes the V.8bis-specific algorithms and is shown in Figure 2-3.

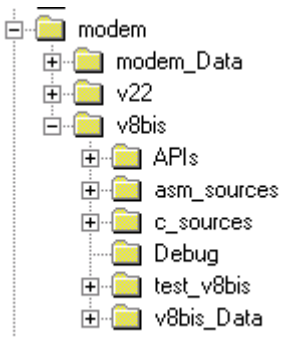

Figure 2-3. *V8bis* Directory Structure

The ***v8bis*** directory includes:

- ***APIs*** - includes both C and assembly API files for the V.8bis library
- ***asm_sources*** - Assembly source files to implement the V.8bis state machine
- ***c_sources*** - C source files to implement the V.8bis state machine

The ***test_v8bis*** directory includes:

- ***Config*** - includes user-configurable files *appconfig.h*, *appconfig.c*, and *linker.cmd* for testing the V.8bis library.
- ***c_sources*** - Test source files in C for both Initiating and Responding stations.

Chapter 3
V.8bis Library Interfaces

3.1 V.8bis Services

V.8bis provides the procedures for identification and selection of common modes of operation between DCEs and DTEs over the public-switched telephone network and on leased point-to-point telephone-type circuits. The communication between the Initiating station and the Responding station is through a set of signals and messages. A *signal* is a combination of dual and single tones, while a *message* is transmitted using V.21 modulation. Apart from certain initialization inputs detailed in **Section 3.3.2**, the library accepts as input the samples received by the user from the remote end. The sampling rate is 7200 samples/sec. During transmission, each station, calls back the user with samples to be transmitted. The user application must transmit these samples to the Codec at 7200 samples/sec. The V.8bis library can be accessed through the APIs: *v8bisCreate*, *v8bisInit*, *v8bisProcess*, and *v8bisDestroy*.

3.2 Interface

The C interface for V.8bis library services is defined in the C header file *v8bis.h*, shown in **Code Example 3-1**:

Code Example 3-1. C Header File *v8bis.h*

```
/* File: v8bis.h */

#ifndef __V8bis_H
#define __V8bis_H

/*
      This include file is the master include file for the
      V8bis protocol. The applications using v8bis should
      include this file
*/

/**************************
 Foundational Include Files
**************************/

#include "port.h"
```

```
/************************
 Flags
 ************************/

 /* These are the flags returned by v8bisProcess()
  * function. Please refer Sec. 3.1.1.1.3 of the
  * V.8bis Library user manual for more details.
  */

 #define V8BIS_BUSY 0
 #define V8BIS_FREE -1


/*****************************************
     V8bis Message Types
*****************************************/

/* These are the inputs the user has to use during
 * the setting up of the V.8bis input buffer.
 * Please refer to:
 *    1. Appendix of the V.8bis Library User Manual (Sec. A.1).
 *    2. The test files of V.8bis: test_v8bisIS.c and test_v8bisRS.c
 */

#define  V8BIS_NIL_RX_HOST_MESSAGE            0x0000
#define  V8BIS_CONFIGURATION_MESSAGE          0x0001
#define  V8BIS_CAPABILITIES_MESSAGE           0x0002
#define  V8BIS_PRIORITIES_MESSAGE             0x0003
#define  V8BIS_REMOTE_CAPABILITIES_MESSAGE    0x0004
#define  V8BIS_TX_GAIN_FACTOR_MESSAGE         0x0007


/*******************
 Kind of station
*******************/

/* To be used by the user to configure V.8bis of
 * his end as either Initiating or Responding
 * station. Please refer test_v8bisIS.c or
 * test_v8bisRS.c for more details.
 */

typedef enum
{
      V8BIS_INIT_STATION,
      V8BIS_RESP_STATION,
} v8bis_eStation;


/*********************
 enums for host/user
**********************/
```

```
/* V8bis library returns one of the following messages
 * to the host followed by appropriate data. The host has
 * to use these, to check in his application, what type of
 * message is returned by V.8bis library, to take necessary
 * further action. Please refer to:
 *     1. Appendix of V.8bis Library User Manual (Sec. A.2).
 */

#define V8BIS_NIL_TX_HOST_MESSAGE             0x0000
#define V8BIS_ACK_MESSAGE                     0x0001
#define V8BIS_ERROR_MESSAGE                   0x0002
#define V8BIS_SUCCESS_INITIATE_HANDSHAKE      0x0003
#define V8BIS_SUCCESS_LOOK_FOR_HANDSHAKE      0x0004


/* These are the possible types of errors that the
 * V.8bis library can return to the host, following the
 * "V8BIS_ERROR_MESSAGE" message as defined above. The host
 * has to take appropriate action as per the error.
 */

#define V8BIS_NIL_ID                          0x0000
#define V8BIS_MODE_NOT_SUPPORTED              0x0001
#define V8BIS_RECEIVED_INVALID_MSG            0x0002
#define V8BIS_RECEIVED_NAK1_MSG               0x0003
#define V8BIS_TIMED_OUT                       0x0004
#define V8BIS_TRANSACTION_BEGUN               0x0005
#define V8BIS_INVALID_MSG_FORMAT              0x0006
#define V8BIS_RECEIVED_NAK2_Or_3_MSG          0x0007


/*****************************************
     Structure for V8bis Capability List
*****************************************/

/* Receiver callback structure */

typedef struct
{
      void (*pCallback) (void *pCallbackArg,
                         Word16 *pChars,
                         UWord16 NumberChars);
      void *pCallbackArg;
} v8bis_sRXCallback;


/* Transmitter callback structure */

typedef struct
{
      void (*pCallback) (void *pCallbackArg,
                         Word16 *pSamples,
                         UWord16 NumberSamples);
      void *pCallbackArg;
} v8bis_sTXCallback;
```

```
/* User configurable structure. This is the format
 * in which the user can pass the parameters to V.8bis
 * library. Please see the test files: test_v8bisIS.c
 * and test_v8bisRS.c for more details.
 */

typedef struct
{
      v8bis_eStation Station;        /* Station type */
      UWord16 *MessagePtr;           /* Input buffer pointer to V.8bis */
      v8bis_sTXCallback TXCallback;  /* Tx. Callback structure */
      v8bis_sRXCallback RXCallback;  /* Rx. Callback structure */
} v8bis_sConfigure;


/* V8bis handle structure. This is strictly for V.8bis
 * internal use only.
 */

typedef struct
{
      Word16 *Output;
      v8bis_eStation Station;
      UWord16 *MessagePtr;
      v8bis_sTXCallback *TXCallback;
      v8bis_sRXCallback *RXCallback;
} v8bis_sHandle;

/***************************
 Function Prototypes
***************************/

EXPORT v8bis_sHandle *v8bisCreate (v8bis_sConfigure *pConfig);

EXPORT Result v8bisInit (v8bis_sHandle *pV8bis, v8bis_sConfigure *pConfig);

EXPORT Result v8bisProcess (v8bis_sHandle *pV8bis,
                            Word16 *pSamples,
                            UWord16 NumSamples);

EXPORT void v8bisDestroy (v8bis_sHandle *pV8bis);

#endif
```

3.3 Specifications

The following pages describe the V.8bis library functions.

Function arguments for each routine are described as *in*, *out*, or *inout*. An *in* argument means that the parameter value is an input only to the function. An *out* argument means that the parameter value is an output only from the function. An *inout* argument means that a parameter value is an input to the function, but the same parameter is also an output from the function.

Typically, *inout* parameters are input pointer variables in which the caller passes the address of a pre-allocated data structure to a function. The function stores its results within that data structure. The actual value of the *inout* pointer parameter is not changed.

3.3.1 *v8bisCreate*

Call(s):

```
v8bis_sHandle *v8bisCreate (v8bis_sConfigure *pConfig);
```

Required Header: "v8bis.h"

Arguments:

Table 3-1. *v8bisCreate* Arguments

pConfig	in	Points to the configuration data for V.8bis

Description: The *v8bisCreate* function creates an instance of V.8bis. During the *v8bisCreate* call, any dynamic resources required by the V.8bis algorithm are allocated. In each call to the *v8bisCreate* function, 19 external memory words are allocated. The library allocates memory dynamically using the *mem* library routines shown in **Code Example 3-2**.

Code Example 3-2. *mem* Library

```
#include "v8bis.h"
#include "v8bis_typedef.h"
#include "mem.h"

v8bis_sHandle *v8bisCreate (v8bis_sConfigure *pConfig)
{
       v8bis_sHandle *pV8bis;
       Word16 temp;

       /* Allocate memory for structure */

       pV8bis = (v8bis_sHandle *) memMallocEM (sizeof (v8bis_sHandle));
       if (pV8bis == NULL) return (NULL);

       pV8bis->Output = (Word16 *) memMallocEM (10 * sizeof (Word16));
       if (pV8bis->Output == NULL) return (NULL);

       pV8bis->TXCallback = (v8bis_sTXCallback *) memMallocEM (sizeof
                                 (v8bis_sTXCallback));
       if (pV8bis->TXCallback == NULL) return (NULL);
       pV8bis->RXCallback = (v8bis_sRXCallback *) memMallocEM (sizeof
                                 (v8bis_sRXCallback));
       if (pV8bis->RXCallback == NULL) return (NULL);

       /* Copy configuration into Handle, i.e., initialize them */
       pV8bis->Station = pConfig->Station;
       pV8bis->MessagePtr = pConfig->MessagePtr;
       pV8bis->TXCallback->pCallback = pConfig->TXCallback.pCallback;
       pV8bis->TXCallback->pCallbackArg = pConfig->TXCallback.pCallbackArg;
       pV8bis->RXCallback->pCallback = pConfig->RXCallback.pCallback;
       pV8bis->RXCallback->pCallbackArg = pConfig->RXCallback.pCallbackArg;

       return (pV8bis);
}
```

For details on the *v8bis_sHandle* structure, please refer to **Code Example 3-1**. The *pConfig* argument points to the *v8bis_sConfigure* structure used to configure the V.8bis operation; for details, see **Code Example 3-3**.

If a *v8bisCreate* function is called to create an instance, then *v8bisDestroy* (see **Section 3.3.4**) should be used to destroy the instance.

Alternatively, the user can allocate memory statically which requires duplicating all statements in the *v8bisCreate* function. In this case, the user can call the *v8bisInit* function directly, bypassing the *v8bisCreate* function. If the user dynamically allocates memory without calling *v8bisCreate*, then the user himself must destroy the memory allocated.

Returns: Upon successful completion, the *v8bisCreate* function will return a pointer to the specific instance of V.8bis created. If *v8bisCreate* is unsuccessful for any reason, it will return "NULL".

Special Considerations:

- The current implementation of V.8bis requires memory sections to be allocated via linker commands. For details on linking applications, see **Chapter 5**.
- The current V.8bis implementation is not re-entrant, so multiple instances are not supported.

In **Code Example 3-3**, the application creates an instance of V.8bis.

Code Example 3-3. Use of *v8bisCreate* Interface

```
#include "v8bis.h"

Note: This example only illustrates the Initiating Station (IS). For details on the
Responding Station, see the test_v8bisRS.c file in SDK

/* An example structure for storing the output */
typedef struct
{
    char buffer[50];
    UWord16 offset;
} WriteOutput; /* User-context structure */

/* Function prototypes for Transmit and Receive callbacks */
void ISCallbackTX (void *pCallbackArg, Word16 *pSamples, UWord16  NumSamples);
void ISCallbackRX (void *pCallbackArg, Word16 *pChars, UWord16  NumChars);

void test_v8bisIS (void)
{
     v8bis_sConfigure pConfig; /* Configuration structure */
     v8bis_sHandle *pV8bis; /* Handle structure */
     UWord16 InputIS[ ] = {See Appendix A for filling this buffer};
     WriteOutput MS; /* Structure for writing the v8bis output (mode selected) */

     /* Initialize pConfig */

     pConfig.Station = V8BIS_INIT_STATION;
     pConfig.MessagePtr = &InputIS[0];
     pConfig.TXCallback.pCallback = ISCallbackTX;
     pConfig.TXCallback.pCallbackArg = NULL; /* NULL because, user has to just
                                               transmit the samples given by v8bis, by
                                               writing into the codec. */
     pConfig.RXCallback.pCallback = ISCallbackRX;
```

```
    pConfig.RXCallback.pCallbackArg = (WriteOutput *) (&MS); /* Mode selected is
                                                                provided by v8bis,
                                                                which has to be used
                                                                by the user for
                                                                further processing */

    pV8bis = v8bisCreate (&pConfig);  /* Create an instance of V.8bis*/
    ....
}
```

For details on structures used in the above example, see **Code Example 3-1**.

3.3.2 *v8bisInit*

Call(s):

```
Result v8bisInit (v8bis_sHandle *pV8bis,
                  v8bis_sConfigure *pConfig);
```

Required Header: "v8bis.h"

Arguments:

Table 3-2. *v8bisInit* Arguments

pV8bis	in	Handle to an instance of V.8bis
pConfig	in	A pointer to a data structure containing data for initializing the V.8bis algorithm

Description: The *v8bisInit* function will initialize the V.8bis algorithm. During initialization, all resources will be set to their initial values in preparation for V.8bis operation. Before calling the *v8bisInit* function, a V.8bis instance must be created. The V.8bis instance (*pV8bis*) can be created either by calling the *v8bisCreate* function (see **Section 3.3.1**), or by statically allocating memory, which does not require a call to the *v8bisCreate* function.

The parameter *pConfig* points to a data structure of type `v8bis_sConfigure`; its fields initialize V.8bis operation in the following manner:

Station Indicates the type of station:

`V8BIS_INIT_STATION` Configures V.8bis as the initiating station

`V8BIS_RESP_STATION` Configures V.8bis as the responding station

MessagePtr A pointer to a buffer of type Uword16. Points to a buffer containing host-config-word, local capabilities, remote capabilities, priorities list, and transmission gain. Details on filling the buffer pointed to by ***this pointer are found in* Appendix A.**

TXCallback A structure of type `v8bis_sTXCallback;` describes the procedure which V.8bis will call as samples are generated by the V.8bis transmitter. The callback procedure has the following declaration:

```
void (*pCallback) (void *pCallbackArg,
                   UWord16 *pSamples,
                   UWord16 NumberSamples);
```

The callback procedure parameter, *pCallbackArg*, is supplied by the user in the *v8bis_sTXCallback* structure; this value is passed back to the user during the call to the callback procedure. Typically, *pCallbackArg* points to context information used by the callback procedure, which must be written by the user. The pointer to the buffer containing samples, *pSamples*, is to be transmitted to the remote station. The total number of samples in the buffer to be transmitted is called *NumberSamples*.

RXCallback A structure of type `v8bis_sRXCallback;` describes the procedure which *v8bis* will call once the transaction under consideration is **complete**. Note that this callback is called by V.8bis **only** at the end. Certain bytes of information are returned to the user, which will indicate the common mode of operation between two stations. To interpret the output octets, refer to Sec. 8 of the V.8bis Standard, Revision 1998.The callback procedure has the following declaration:

```
void (*pCallback) (void *pCallbackArg,
                   UWord16 *pChars,
                   UWord16 NumberChars);
```

The callback procedure parameter, *pCallbackArg*, is supplied by the user in the *v8bis_sRXCallback* structure; this value is passed back to the user during the call to the callback procedure. Typically, *pCallbackArg* points to context information used by the callback procedure, which must be written by the user. The pointer to the character buffer given to the user by his receiver is *pChars* and contains the common mode of transaction selected between the two stations. The number of characters in the buffer pointed to by *pChars* is called *NumberChars*.

Returns: This function always returns “PASS”.

Special Considerations: None

Code Example 3-4. Use of *v8bisInit* Interface

```
#include “v8bis.h”

Note: This example only illustrates the Initiating Station (IS). For details on the
Responding Station, please see the test_v8bisRS.c file in SDK

/* An example structure for storing the output */
typedef struct
{
    char buffer[50];
    UWord16 offset;
} WriteOutput;             /* User-context structure */

void test_v8bisIS (void)
{
     Result res;
     v8bis_sConfigure pConfig; /* Configuration structure */
     v8bis_sHandle *pV8bis; /* Handle structure */
     UWord16 InputIS[ ] = {See Appendix A for filling this buffer};
     WriteOutput MS; /* Structure for writing the v8bis output (mode selected) */

     MS.offset = 0;
     /* Initialize pConfig */

     pConfig.Station = V8BIS_INIT_STATION;
     pConfig.MessagePtr = &InputIS[testcase_no][0];
     pConfig.TXCallback.pCallback = ISCallbackTX;
     pConfig.TXCallback.pCallbackArg = NULL; /* NULL because, user has to just
                                               transmit the samples given by v8bis, by
                                               writing into the codec. */
     pConfig.RXCallback.pCallback = ISCallbackRX;
     pConfig.RXCallback.pCallbackArg = (WriteOutput *) (&MS); /* Mode selected is
                                                                 provided by v8bis,
                                                                 which has to be used
                                                                 by the user for
                                                                 further processing */

     pV8bis = v8bisCreate (&pConfig);  /* Create an instance of V.8bis */
     ....
     v8bisInit (pV8bis, &pConfig);     /* V8bis Initialization */
     ....
}
```

For details on structures used in the above example, see **Code Example 3-1**.

3.3.3 *v8bisProcess*

Call(s):

```
Result v8bisProcess (v8bis_sHandle *pV8bis,
                     Word16 *pSamples,
                     UWord16 NumSamples);
```

Required Header: "v8bis.h"

Arguments:

Table 3-3. *v8bisProcess* Arguments

pV8bis	in	Handle to an instance of V.8bis
pSamples	in	Pointer to the user-given Codec samples received from the remote station in 16-bit, 1.15 format, linear PCM to be processed by V.8bis in the local station
NumSamples	in	The number of samples words to be processed

Description: This function processes the signal/message samples received by the user from the remote station, as required by the V.8bis state machine shown in the V.8bis standard. This function must be called in a loop as illustrated in **Code Example 3-5**, making a number of references to *v8bisProcess* function. The parameter *pV8bis* must have been generated from a call to the *v8bisCreate* function.

Returns: This function returns "V8BIS_BUSY" if the transactions are not complete; otherwise, it returns "V8BIS_FREE".

Code Example 3-5. Use of *v8bisProcess* Interface

```
#include "v8bis.h"

Note: This example only illustrates the Initiating Station (IS). For details on the
Responding Station, please see the test_v8bisRS.c file in SDK

/* An example structure for storing the output */
typedef struct
{
    char buffer[50];
    UWord16 offset;
} WriteOutput;             /* User-context structure */

void test_v8bisIS (void)
{
     int i;
     UInt16 j;
     Result res;
     v8bis_sConfigure pConfig; /* Configuration structure */
     v8bis_sHandle *pV8bis; /* Handle structure */
     UWord16 InputIS[ ] = {See Appendix A for filling this buffer};
     WriteOutput MS; /* Structure for writing the v8bis output (mode selected) */

     MS.offset = 0;
     /* Initialize pConfig */
```

```
     pConfig.Station = V8BIS_INIT_STATION;
     pConfig.MessagePtr = &InputIS[testcase_no][0];
     pConfig.TXCallback.pCallback = ISCallbackTX;
     pConfig.TXCallback.pCallbackArg = NULL; /* NULL because, user has to just
                                               transmit the samples given by v8bis, by
                                               writing into the codec. */
     pConfig.RXCallback.pCallback = ISCallbackRX;
     pConfig.RXCallback.pCallbackArg = (WriteOutput *) (&MS); /* Mode selected is
                                                                 provided by v8bis,
                                                                 which has to be used
                                                                 by the user for
                                                                 further processing */

     /* Call APIs */
     pV8bis = v8bisCreate (&pConfig);  /* Create an instance of V.8bis */
     ....
     v8bisInit (pV8bis, &pConfig);     /* V8bis Initialization */
     ....

     /* Process the received samples until the transaction is
     * complete (passed or failed) */

     while (res == V8BIS_BUSY)
     {
          ....
          /* Collect NUMRX_SAMPLES number of samples from the codec into CodecRxBuffer
           * buffer. */
          ....

          /* Process the received samples. Once the processing is over v8bisProcess
           * function will return V8BIS_FREE flag. This is the condition for exiting
           * this while loop. */

          res = v8bisProcess (pV8bis, CodecRxBuffer, NUMRX_SAMPLES);
     }

     ....
}
```

For details on structures used in the above example, see **Code Example 3-1**.

3.3.4 *v8bisDestroy*

Call(s):

```
void v8bisDestroy (v8bis_sHandle *pV8bis);
```

Required Header: "v8bis.h"

Arguments:

Table 3-4. *v8bisDestroy* Arguments

pV8bis	in	Handle to an instance of V.8bis generated by a call to *v8bisCreate*

Description: The *v8bisDestroy* function destroys the instance of V.8bis originally created by a call to *v8bisCreate*. If an instance was created by the user himself **without** using the *v8bisCreate* function, the user must free the memory allocated.

Returns: None

Code Example 3-6. Use of *v8bisDestroy* Interface

```
#include "v8bis.h"

Note: This example only illustrates the Initiating Station (IS). For details on the
Responding Station, please see the test_v8bisRS.c file in SDK

/* An example structure for storing the output */
typedef struct
{
    char buffer[50];
    UWord16 offset;
} WriteOutput;              /* User-context structure */

void test_v8bisIS (void)
{
     int i;
     UInt16 j;
     Result res;
     v8bis_sConfigure pConfig; /* Configuration structure */
     v8bis_sHandle *pV8bis; /* Handle structure */
     UWord16 InputIS[ ] = {See Appendix A for filling this buffer};
     WriteOutput MS; /* Structure for writing the v8bis output (mode selected) */

     MS.offset = 0;
     /* Initialize pConfig */

     pConfig.Station = V8BIS_INIT_STATION;
     pConfig.MessagePtr = &InputIS[testcase_no][0];
     pConfig.TXCallback.pCallback = ISCallbackTX;
     pConfig.TXCallback.pCallbackArg = NULL; /* NULL because, user has to just
                                               transmit the samples given by v8bis, by
                                               writing into the codec. */
     pConfig.RXCallback.pCallback = ISCallbackRX;
```

```
     pConfig.RXCallback.pCallbackArg = (WriteOutput *) (&MS); /* Mode selected is
                                                                 provided by v8bis,
                                                                 which has to be used
                                                                 by the user for
                                                                 further processing */


     /* Call APIs */
     pV8bis = v8bisCreate (&pConfig);  /* Create an instance of V.8bis */
     ....
     v8bisInit (pV8bis, &pConfig);     /* V8bis Initialization */
     ....

     /* Process the received samples until the transaction is
     * complete (passed or failed) */

     while (res == V8BIS_BUSY)
     {
          ....
          /* Collect NUMRX_SAMPLES number of samples from the codec into CodecRxBuffer
           * buffer. */
          ....

          /* Process the received samples. Once the processing is over v8bisProcess
           * function will return V*BIS_FREE flag. This is the condition for exiting
           * this while loop. */

          res = v8bisProcess (pV8bis, CodecRxBuffer, NUMRX_SAMPLES);
     }

     /* Destroy the v8bis instance */

     v8bisDestroy (pV8bis);

     ....
}
```

For details on structures used in the above example, see **Code Example 3-1**.

Chapter 4
Building the V.8bis Library

4.1 Building the V.8bis Library

The V.8bis library combines all of the components described in previous sections into one library: ***v8bis.lib***. To build this library, a Metrowerks' CodeWarrior project, ***v8bis.mcp***, is provided. This project and all the necessary components to build the V.8bis library are located in the ***...modem\v8bis*** directory of the SDK directory structure.

There are two methods to execute a system library project build: dependency build and direct build.

4.1.1 Dependency Build

Dependency build is the easiest approach and requires no additional work on the user's part. If you add the V.8bis library project, ***v8bis.mcp***, to your application project as shown in **Figure 4-1**, the V.8bis library will automatically build when the application is built.

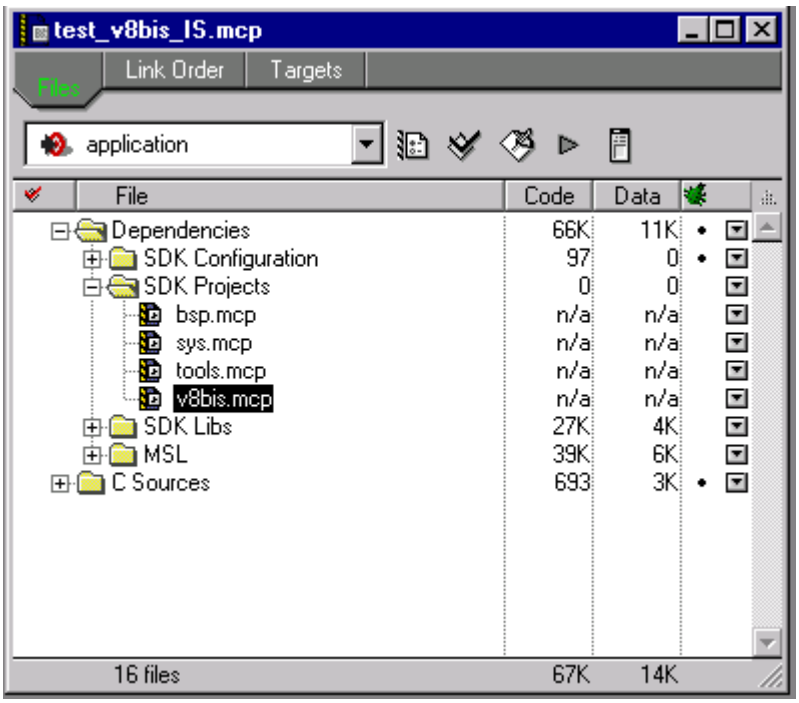

Figure 4-1. Dependency Build for *v8bis* Project

4.1.2 Direct Build

Direct build allows you to build a V.8bis library independently of any other build. Follow these steps:

Step 1. Open *v8bis.mcp* project, as shown in **Figure 4-2**:

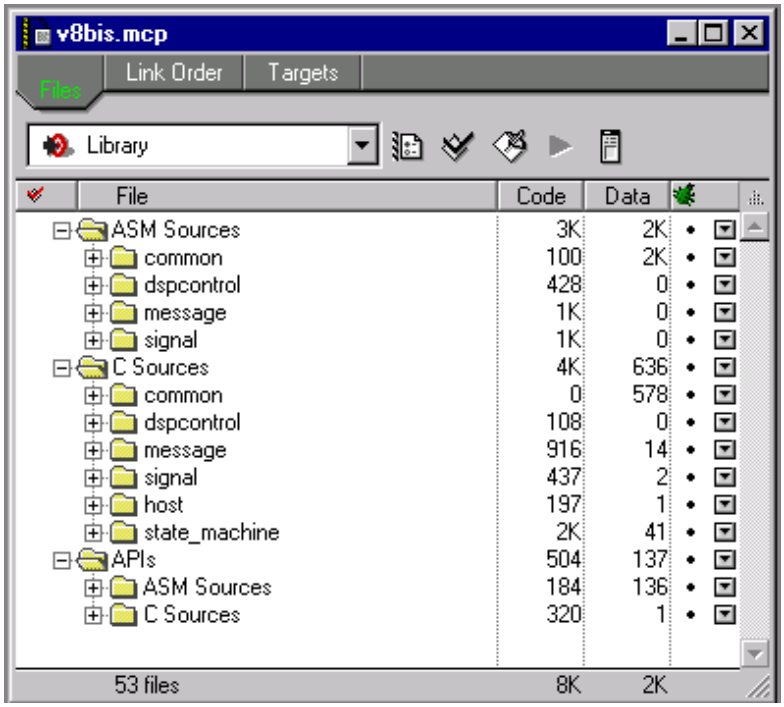

Figure 4-2. *v8bis.mcp* Project

Step 2. Execute the build by pressing function key [F7] or by choosing *Make* from the Project menu; see **Figure 4-3**.

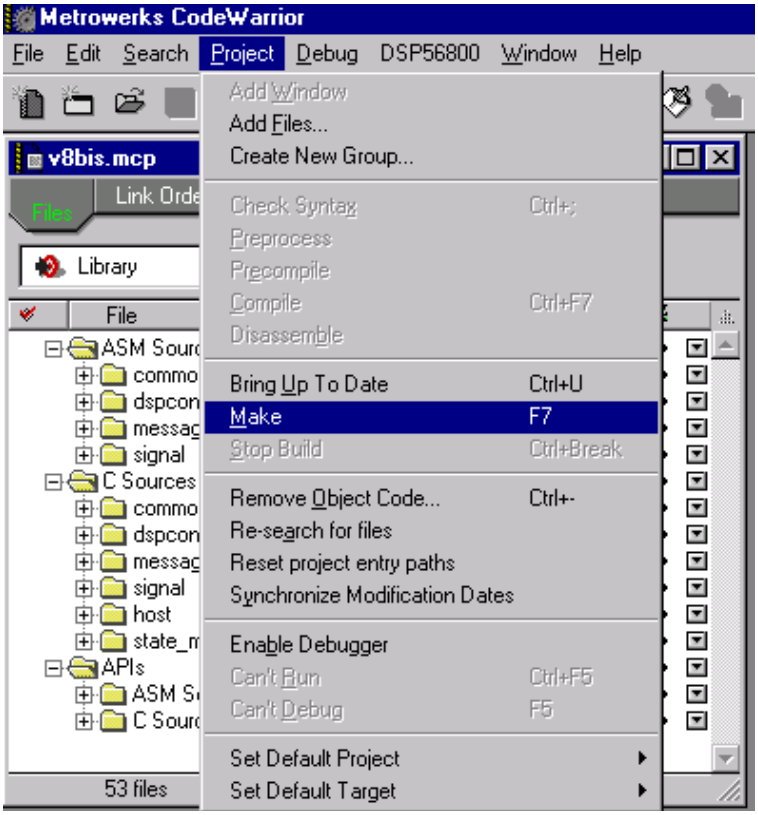

Figure 4-3. Execute Make

At this point, if the build is successful, the *v8bis.lib* library file is created in the ***...\modem\v8bis\Debug*** directory.

Chapter 5
Linking Applications with the V.8bis Library

5.1 V.8bis Library

The V.8bis library consists of the V.8bis state machine as specified in the V.8bis standard. The instance of V.8bis can be created and initialized by the *v8bisCreate* function, detailed in **Section 3.3.1**. The library contains APIs, which provide the interface between the user application and the V.8bis modules. To use the V.8bis library, APIs must be called in the following order:

— v8bisCreate (.....);

— v8bisInit (.....);

— v8bisProcess (.....); /* The user needs to call this module whenever there are samples to be transmitted, or when samples received are to be processed */

— v8bisDestroy (.....);

5.1.1 Library Sections

The internal data memory requirement for V.8bis is zero.
The external data memory sections for V.8bis are found in the example at the end of this section.

Note: All sections are grouped together for placing in different memory regions on the target. The memory regions and the sections under each region that follow are for reference. The ORIGIN and LENGTH fields of the memory regions can be changed by the user application, but ensure that the regions do not overlap.

An example *linker.cmd* file is shown in **Code Example 5-1**.

Code Example 5-1. *linker.cmd* File

```
# Linker.cmd file for DSP56824EVM External RAM
# using both internal and external data memory (EX = 0)
# and using external program memory (Mode = 3)

MEMORY {
       .pram   (RWX) : ORIGIN = 0x0000, LENGTH = 0xFF80  # ? external program memory
       .avail  (RW)  : ORIGIN = 0x0000, LENGTH = 0x0030  # available
```

```
        .cwregs (RW)  : ORIGIN = 0x0030, LENGTH = 0x0010  # C temp registrs in
                                CodeWarrior
        .im1    (RW)  : ORIGIN = 0x0040, LENGTH = 0x07C0  # data 1
        .rom    (R)   : ORIGIN = 0x0800, LENGTH = 0x0800  # internal data ROM
        .im2    (RW)  : ORIGIN = 0x1000, LENGTH = 0x0600  # data 2
        .hole   (R)   : ORIGIN = 0x1600, LENGTH = 0x0A00  # hole
        .data   (RW)  : ORIGIN = 0x2000, LENGTH = 0xC000  # data segment
        .em     (RW)  : ORIGIN = 0xE000, LENGTH = 0x1000  # data 3
        .stack  (RW)  : ORIGIN = 0xF000, LENGTH = 0x0F80  # stack
        .onchip1(RW)  : ORIGIN = 0xFF80, LENGTH = 0x0040  # on-chip peripheral
                                registers
        .onchip2(RW)  : ORIGIN = 0xFFC0, LENGTH = 0x0040  # on-chip peripheral
                                registers

        # V.8bis specific memory regions
        #----------------------------
        .V8bis_align_ext_data (RW)  : ORIGIN = 0x4000, LENGTH = 0x0600
}

FORCE_ACTIVE {FconfigInterruptVector}

SECTIONS {
        #
        # Data (X) Memory Layout
        #
        _EX_BIT     = 0;
        # Internal Memory Partitions (for mem.h partitions)
        _NUM_IM_PARTITIONS = 2;  # .im1 and .im2
        # External Memory Partition (for mem.h partitions)
        _NUM_EM_PARTITIONS = 1;   # .em

        .main_application_code :
        {
                # .text sections

                #  config.c MUST be placed first, otherwise the Interrupt Vector
                #  configInterruptVector will not be located at the correct address,
                # P:0x0000

                config.c (.text)
                * (.text)
                * (rtlib.text)
                * (fp_engine.text)
                * (user.text)

        } > .pram

        .main_application_data :
        {
                #
                # Define variables for C initialization code
                #
                F_Xdata_start_addr_in_ROM = ADDR(.rom) + SIZEOF(.rom) / 2;
                F_StackAddr               = ADDR(.stack);
                F_StackEndAddr            = ADDR(.stack) + SIZEOF(.stack) / 2  - 1;
                F_Xdata_start_addr_in_RAM = .;
```

```
        #
        # Memory layout data for SDK INCLUDE_MEMORY (mem.h) support
        #
        FmemEXbit = .;
              WRITEH(_EX_BIT);
        FmemNumIMpartitions = .;
              WRITEH(_NUM_IM_PARTITIONS);
        FmemNumEMpartitions = .;
              WRITEH(_NUM_EM_PARTITIONS);
        FmemIMpartitionList = .;
              WRITEH(ADDR(.im1));
              WRITEH(SIZEOF(.im1) / 2);
              WRITEH(ADDR(.im2));
              WRITEH(SIZEOF(.im2) / 2);
        FmemEMpartitionList = .;
              WRITEH(ADDR(.em));
              WRITEH(SIZEOF(.em) /2);

        # .data sections
        * (.data)
        * (fp_state.data)
        * (rtlib.data)

        # V8bis data sections start here
        #------------------------------
        * (v21_xrom.data)
        * (dtmf_rom.data)
        .=ALIGN(64);
        * (v21_mod_ram.data)
        .=ALIGN(256);
        * (v21_prom1.data)
        .=ALIGN(512);
        * (v21_prom2.data)
        # V8bis data sections end here
        #----------------------------

        F_Xdata_ROMtoRAM_length = 0;
        F_bss_start_addr = .;
        _BSS_ADDR = .;
        * (rtlib.bss.lo)
        * (.bss)

        # V8bis bss sections start here
        #------------------------------
        * (ToneGen_Common_Variable.bss)
        * (V8BIS_IS_RS_INIT.bss)
        * (HOST.bss)
        .=ALIGN(32);
        * (ToneDet_Common_Variable.bss)
        # V8bis bss sections end here
        #----------------------------

        F_bss_length = . - _BSS_ADDR;  # Copy DATA
    } > .data

    FArchIO   = ADDR(.onchip2);
```

```
        # V8bis data section starts here. This section cannot be
        # included in .data region, since inclusion leads to memory
        # overlap. This could be a tool bug
        #-------------------------------------------------------

        .V8bis_align_ext_data :
        {
                .=ALIGN(512);
                * (V8bis_Codec.data)
        } > .V8bis_align_ext_data

        # V8bis data section ends here
        #----------------------------
}
```

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

Chapter 6
V.8bis Applications

6.1 Test and Demo Applications

To verify the V.8bis algorithm, test and demo applications have been developed. Refer to the Targeting Motorola DSP568xx Platform manual for the DSP you are using to see if the test and demo applications are available for your target.

Chapter 7
License

7.1 Limited Use License Agreement

LIMITED USE LICENSE AGREEMENT

PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING THIS SOFTWARE. BY USING OR COPYING THE SOFTWARE, YOU AGREE TO THE TERMS OF THIS AGREEMENT.

The software in either source code form ("Source") or object code form ("Object") (cumulatively hereinafter "Software") is provided under a license agreement ("Agreement") as described herein. Any use of the Software including copying, modifying, or installing the Software so that it is usable by or accessible by a central processing unit constitutes acceptance of the terms of the Agreement by the person or persons making such use or, if employed, the employer thereof ("Licensee") and if employed, the person(s) making such use hereby warrants that they have the authority of their employer to enter this license agreement,. If Licensee does not agree with and accept the terms of this Agreement, Licensee must return or destroy any media containing the Software or materials related thereto, and destroy all copies of the Software.

The Software is licensed to Licensee by Motorola Incorporated ("Motorola") for use under the terms of this Agreement. Motorola retains ownership of the Software. Motorola grants only the rights specifically granted in this Agreement and grants no other rights. Title to the Software, all copies thereof and all rights therein, including all rights in any intellectual property including patents, copyrights, and trade secrets applicable thereto, shall remain vested in Motorola.

For the Source, Motorola grants Licensee a personal, non-exclusive, non-assignable, revocable, royalty-free right to use, copy, and make derivatives of the Source solely in a development system environment in order to produce object code solely for operating on a Motorola semiconductor device having a central processing unit ("Derivative Object").

For the Object and Derivative Object, Motorola grants Licensee a personal, non-exclusive, non-assignable, revocable, royalty-free right to copy, use, and distribute the Object and the Derivative Object solely for operating on a Motorola semiconductor device having a central processing unit.

Licensee agrees to: (a) not use, modify, or copy the Software except as expressly provided herein, (b) not distribute, disclose, transfer, sell, assign, rent, lease, or otherwise make available the Software, any derivatives thereof, or this license to a third party except as expressly provided herein, (c) not remove obliterate, or otherwise defeat any copyright, trademark, patent or proprietary notices, related to the Software (d) not in any form export, re-export, resell, ship or divert or cause to be exported, re-exported, resold, shipped, or diverted, directly or indirectly, the Software or a direct product thereof to any country which the United States government or any agency thereof at the time of export or re-export requires an export license or other government approval without first obtaining such license or approval.

THE SOFTWARE IS PROVIDED ON AN "AS IS" BASIS AND WITHOUT WARRANTY OF ANY KIND INCLUDING (WITHOUT LIMITATION) ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL MOTOROLA BE LIABLE FOR ANY LIABILITY OR DAMAGES OF ANY KIND INCLUDING, WITHOUT LIMITATION, DIRECT OR INDIRECT OR INCIDENTAL OR CONSEQUENTIAL OR PUNITIVE DAMAGES OR LOST PROFITS OR LOSS OF USE ARISING FROM USE OF THE SOFTWARE OR THE PRODUCT REGARDLESS OF THE FORM OF ACTION OR THEORY OF LIABILITY (INCLUDING WITHOUT LIMITATION, ACTION IN CONTRACT, NEGLIGENCE, OR PRODUCT LIABILITY) EVEN IF MOTOROLA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THIS DISCLAIMER OF WARRANTY EXTENDS TO LICENSEE OR USERS OF PRODUCTS AND IS IN LIEU OF ALL WARRANTIES WHETHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR PARTICULAR PURPOSE.

Motorola does not represent or warrant that the Software is free of infringement of any third party patents, copyrights, trade secrets, or other intellectual property rights or that Motorola has the right to grant the licenses contained herein. Motorola does not represent or warrant that the Software is free of defect, or that it meets any particular requirements or need of the Licensee, or that it conforms to any documentation, or that it meets any standards.

Motorola shall not be responsible to maintain the Software, provide upgrades to the Software, or provide any field service of the Software. Motorola reserves the right to make changes to the Software without further notice to Licensee.

The Software is not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Software could create a situation where personal injury or death may occur. Should Licensee purchase or use the Software for any such unintended or unauthorized application, Licensee shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the Software.

The term of this Agreement is for as long as Licensee uses the Software for its intended purpose and is not in default of any provisions of this Agreement. Motorola may terminate this Agreement if Licensee is in default of any of the terms and conditions of this Agreement.

This Agreement shall be governed by and construed in accordance with the laws of the State of Arizona and can only be modified in a writing signed by both parties. Licensee agrees to jurisdiction and venue in the State of Arizona.

By using, modifying, installing, compiling, or copying the Software, Licensee acknowledges that this Agreement has been read and understood and agrees to be bound by its terms and conditions. Licensee agrees that this Agreement is the complete and exclusive statement of the agreement between Licensee and Motorola and supersedes any earlier proposal or prior arrangement, whether oral or written, and any other communications relative to the subject matter of this Agreement.

Appendix A
Guidelines for Setting Up the Input Buffer

A.1 Inputs to V.8bis

A.1.1 Format of Input Buffer

The input buffer is a combination of host message type and host message data. The format is shown in **Figure A-1**. Also see the test files *test_v8bisIS.c* and *test_v8bisRS.c*. To find these files, go to: ***...\modem\v8bis\test_v8bis\c_sources\.***, and the InputIS[ ] and InputRS[ ] buffers in these files.

Element number in the input buffer	Content	Format i = Information bit x = Don't care
0	V8BIS_CONFIGURATION_MESSAGE	See **Table A-1**
1	The host config word	xxxi iiii iiii iiii
2	V8BIS_TX_GAIN_FACTOR_MESSAGE	See **Table A-1**
3	The gain value	iiii iiii iiii iiii (1.15 format)
4	V8BIS_CAPABILITIES_MESSAGE	See **Table A-1**
5	No. of words in the local capabilities list (**say, N**)	0000 0000 000i iiii
6	First word in local capabilities list	0000 0000 iiii iiii
7	Second word in local capabilities list	0000 0000 iiii iiii
8	...	
5+N	Last word in local capabilities list	0000 0000 iiii iiii
5+N+1	V8BIS_REMOTE_CAPABILITIES_MESSAGE	See **Table A-1**
5+N+2	Number of words in the remote capabilities list (**say, M**)	0000 0000 000i iiii
5+N+3	First word in remote capabilities list	0000 0000 iiii iiii
5+N+4	...	
5+N+M+2	Last word in remote capabilities list	0000 0000 iiii iiii
5+N+M+3	V8BIS_PRIORITIES_MESSAGE	See **Table A-1**

Element number in the input buffer	Content	Format i = Information bit x = Don't care
5+N+M+4	Number of words in the priorities list Example: P. P = (N-1)*8 for initiating station, and P = (M-1)*8 for responding station	0000 0000 iiii iiii
5+N+M+5	First word in the priorities list	0000 0000 iiii iiii
5+N+M+6	...	
5+N+M+P+4	Last word in the priorities list	0000 0000 iiii iiii
	Any number of zeros required can be entered	0000 0000 0000 0000

Figure A-1. Input Buffer Format for V.8bis

The host message type and data are shown in **Table A-1**.

Table A-1. Host Message Types and Data

MESSAGE TYPE		DATA
V8BIS_NIL_RX_HOST_MESSAGE	0000	NULL
V8BIS_CONFIGURATION_MESSAGE	0001	Host Configuration word from **Table**
V8BIS_CAPABILITIES_MESSAGE	0002	As shown in **Table A-11**
V8BIS_PRIORITIES_MESSAGE	0003	As shown in **Table A-12**
V8BIS_REMOTE_CAPABILITIES_MESSAGE	0004	As shown in **Table A-11**
V8BIS_TX_GAIN_FACTOR_MESSAGE	0007	gain(0-1)

A.1.2 Host-Config word

The host config word allows the user to configure certain parameters. It is a 16-bit word; bit pattern and the function of each bit are shown in **Figure A-2**.

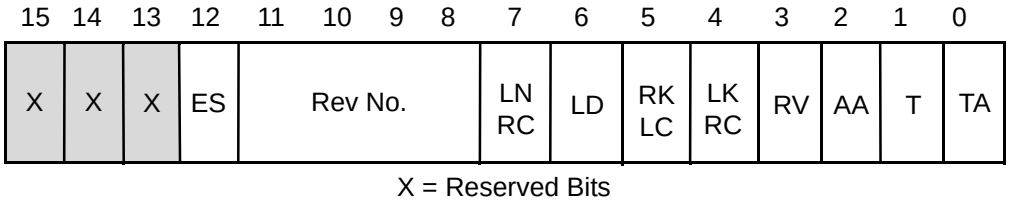

Figure A-2. Configuration Word

TA bit, Table A-2: This bit decides whether the Transmit Ack1 bit should be set or reset in the Identification field's NPAR1, which in turn will decide if the remote modem should transmit ACK1 on receiving the MS message.

Table A-2. TA Bit Function

TA Bit	Function
0	Don't expect ACK1 from remote for MS message
1	Expect ACK1 from remote for MS message

T bit, Table A-3: Decides if the DCE supports telephony mode.

Table A-3. T Bit Function

T Bit	Function
0	Telephony mode not supported
1	Telephony mode supported

AA bit, Table A-4: Turns on/off the local DCE's autoanswering capability.

Table A-4. AA Bit

AA Bit	Function
0	Disables auto answering
1	Enables auto answering

RV bit, Table A-5: Determines if the local DCE knows the remote has V.8bis capability. This in turn determines the type of transaction which the local DCE will take up

Table A-5. RV Bit

RV Bit	Function
0	No knowledge of the remote
1	Knows that remote has V.8bis capability

LKRC bit, Table A-6:This bit decides if the local knows remote capability. If this bit is set, then there will not be a phase of exchanging capabilities during the transaction, minimizing the time taken for choosing the common mode.

Table A-6. LKRC Bit

LKRC Bit	Function
0	Local doesn't know remote's capability
1	Local knows remote's capability

RKLC bit, Table A-7: This bit decides if the remote knows the local capabilities. If this bit is set, then there will not be a phase of exchanging capabilities during the transaction, minimizing the time taken for choosing the common mode

Table A-7. RKLC Bit

RKLC Bit	Function
0	Remote doesn't know local capability
1	Remote knows local capability

LD bit, Table A-8: This bit decides if the local wants the final say in selecting the mode. This will influence the transaction in such a way that the local will finally send out an MS message

Table A-8. LD Bit

LD Bit	Function
0	Local doesn't care
1	Local would like to have the final say in mode selection

LNRC bit, Table A-9: This bit decides if the local will send out a CL message asking for remote's capability.

Table A-9. LNRC Bit

LNRC Bit	Function
0	Local doesn't want remote's capability
1	Local wants to know the remote's capability

Rev No bits:This set of four bits decides the revision number of the V.8bis recommendation that this software has implemented. ***This should always be 0001 for this version***.

ES bit, **Table A-10**: This bit tells the V.8bis software if the telephone line is expected to have echo suppressor.

Table A-10. ES Bit

ES bit	Function
0	No echo suppressor in the telephone network
1	Echo suppressor present in the telephone network

Reserved bits: The reserved bits should be written as zero.

A.1.3 Setting Up the Capabilities List

Local and remote capabilities are set as shown in **Table A-11**. (ID - Identification, SI - Standard Information field). Please refer Sec. 8 of [4]

Table A-11. Capabilities List

0009	Number of words, excluding this word
0012	Revision number & Message type, CL
0083	NPAR1 of ID field, delimit bit, v.8, short v.8
0080	SPAR1 of ID field; only delimit bit. Note: Since there is no parameter set in this field, there will be no NPAR2 byte
0080	NPAR1 of SI field; only delimit bit
00a1	SPAR1 of SI field; delimit bit, analog telephony, data
0002	1st octet of NPAR2 (of DATA) of SI field; V.42
0000	2nd octet of NPAR2 (of DATA) of SI field; None selected
00c2	3rd octet of NPAR2 (of DATA) of SI field; delimit bits and V.22bis
00c2	1st octet of NPAR2 (of analog telephony) of SI field; delimit bits and Audio recording device

ID = Identification
SI = Standard Information field

For details, refer to Section 8 of the V.8bis Standard, Revision 1996

A.1.4 Setting Up Priorities List

Priorities are mentioned in a matrix. Normally, 8 octets describe the priorities of a single octet in the capabilities list. There is no corresponding priority matrix for the Revision Number and Message type. The first entry of the capabilities list should shown the number of octets. **Table A-12** shows an example priority for the SPAR1 of the SI field.

Table A-12. Priority List

0040 (Hex)	Number of words, excluding this word
0001	1st bit is the top priority (DATA application)
0020	6th bit is the second priority (Analog Telephony)
0000	No other Priorities
*	*
0000	No other Priorities
0000	No other Priorities
0000	No other Priorities(63rd word)
0000	No other Priorities(64th word)

A.2 Output from V.8bis

The V.8 bis software returns the negotiated mode of communication to the user as output. This will have certain octets; the first octet will be any specified in **Table A-13**.

Table A-13. Output Message Type and Data

MESSAGE TYPE		DATA
V8BIS_NIL_TX_HOST_MESSAGE	0000	NULL
V8BIS_ACK_MESSAGE	0001	NULL
V8BIS_ERROR_MESSAGE	0002	error_id; see **Table A-14**
V8BIS_SUCCESS_INITIATE_HANDSHAKE	0003	The data will be the common mode of agreement. This message type means that V.8bis is successful, asking the user to initiate the application handshake.
V8BIS_SUCCESS_LOOK_FOR_HANDSHAKE	0004	The data will be the common mode of agreement. This message type means that V.8bis is successful, asking the user to look for the application handshake.

Table A-14. Error Ids

Error_id	
V8BIS_NIL_ID	0000
V8BIS_MODE_NOT_SUPPORTED	0001
V8BIS_RECEIVED_INVALID_MSG	0002
V8BIS_RECEIVED_NAK1_MSG	0003
V8BIS_TIMED_OUT	0004
V8BIS_TRANSACTION_BEGUN	0005
V8BIS_INVALID_MSG_FORMAT	0006
V8BIS_RECEIVED_NAK2_Or_3_MSG	0007

Index

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

How to reach us:
USA/EUROPE/Locations Not Listed: Motorola Literature Distribution; P.O. Box 5405, Denver, Colorado 80217. 1–303–675–2140 or 1–800–441–2447

JAPAN: Motorola Japan Ltd.; SPS, Technical Information Center, 3–20–1, Minami–Azabu. Minato–ku, Tokyo 106–8573 Japan. 81–3–3440–3569

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong. 852–26668334

Technical Information Center: 1–800–521–6274

HOME PAGE: http://www.motorola.com/semiconductors/